Universidad Andrés Bello
Facultad de Ingeniería
Ingeniería en Computación e Informática

Estructura de Datos
Solemne I

Nombre: _____ **Nota:** _____

Profesores: Carlos Contreras Bolton – José Luis Allende – Felipe Reyes González
Ayudantes: Carlos Rey – Daniela Ubilla – Tamara Saéz

Fecha: 22 de Abril del 2014

Instrucciones:

- Coloque su nombre a todas las hojas.
- Apague o silencie sus celulares. **NO** se podrá contestar llamadas ni visualizar el celular. Si se realiza una de las acciones anteriores obtendrá nota mínima.
- Toda copia o intento de copia será calificada con nota mínima.
- Tiene 90 minutos para realizar la prueba.

1	15
2	15
3	15
4	15
5	15
Suma:	75
	+10
Nota:	85

Pregunta 1 (15 puntos)

Realizar una función que imprima una lista de manera inversa usando recursividad y luego quite la recursividad usando pilas.

```
1  #include "lista.h"
2  #include "pila.h"
3
4  void imprimir_lista_inversa(Nodo *i)
5  {
6    if(i != NULL)
7      {
8        imprimir_lista_inversa(i->siguiente);
9        printf("%d ", i->datos->dato1);
10     }
11 }
12
13 void imprimir_lista_inversa_pila(Nodo *i, Pila *p)
14 {
15   for(; i != NULL; i = i->siguiente)
16     apilar(p, i->datos->dato1);
17
18   while(!vacia(p))
19     {
20       printf("%d ", tope(p)->datos->dato1);
21       desapilar(p);
22     }
23   printf("\n");
24 }
25
26 int main(int argc, char *argv[])
27 {
28   Lista *L = creaLista();
29   Pila *P = creaPila();
30   insertar(L, 1);
31   insertar(L, 2);
32   insertar(L, 3);
33   recorrer(L);
34   imprimir_lista_inversa(L->inicio);
35   printf("\n");
36   imprimir_lista_inversa_pila(L->inicio, P);
37   destruirLista(L);
38   destruirPila(P);
39   return 0;
40 }
```

Pregunta 2 (15 puntos)

Realizar una función para calcular el número combinatorio de la ecuación 1 y calcule su eficiencia. El factorial debe ser recursivo.

$$\binom{n}{k} = \frac{n!}{m!(n-m)!}, \qquad n, m \geq 0 \text{ y } n \geq m \tag{1}$$

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  long double factorial(unsigned n)
5  {
6    if(n == 0 || n == 1)
7      return 1;
8    else
9      return n * factorial(n-1);
10 }
11
12 long double numCombinatorio(unsigned n, unsigned m)
13 {
14   return factorial(n) / (factorial(m) * factorial(n - m));
15 }
16
17 int main(int argc, char *argv[])
18 {
19   int n = atoi(argv[1]);
20   int m = atoi(argv[2]);
21   printf("%0.llf\n", numCombinatorio(n, m));
22   return 0;
23 }
```

Estudiamos la eficiencia de la función `factorial`. Su ecuación de recurrencia es:

$$T(n) = \begin{cases} 1 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ T(n-1) + 1 & \text{si } n > 1 \end{cases}$$

Aplicamos expansión:

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 2 \\ &= T(n-3) + 3 \\ &\vdots \\ &= T(n-k) + k \\ &\vdots \ {\scriptstyle k=n-1} \\ &= T(n-(n-1)) + n - 1 \\ &= T(1) + n - 1 \\ &= 1 + n - 1 = n \\ &= O(n) \end{aligned}$$

Por tanto, para estudiar la eficiencia de la función `numCombinatorio` basta con observar que hay tres llamados de la función `factorial` con tamaños n, m y $n - m$. Por tanto, la eficiencia de la función `numCombinatorio` es $n + n - m + m = O(n)$.

Pregunta 3 (15 puntos)

Realizar una función que reciba dos listas y ésta retorne la diferencia entre las listas, es decir, todos los elementos de la primera lista que **NO** están en la segunda.

```
1  #include "lista.h"
2
3  Nodo *localizar(Lista *lista, int dato)
4  {
5    Nodo *i;
6    for(i = lista->inicio; i != NULL; i = i->siguiente)
7      if(i->datos->dato1 == dato)
8        return i;
9    return i;
10 }
11
12 Lista *diferencia(Lista *l1, Lista *l2)
13 {
14   Nodo *aux=l1->inicio;
15   Nodo *tmp;
16   while(aux->siguiente != NULL)
17     {
18       /* busca si el elemento de L1 está en L2 */
19       tmp = localizar(l2, aux->siguiente->datos->dato1);
20       if(tmp != NULL)
21         aux->siguiente = aux->siguiente->siguiente; /* elimina el nodo de la primera lista */
22       else
23         aux = aux->siguiente; /* sino pregunta por el siguiente elemento */
24     }
25   return l1;
26 }
27
28 int main(int argc, char *argv[])
29 {
30   Lista *L1 = creaLista();
31   Lista *L2 = creaLista();
32   insertar(L1, 1);
33   insertar(L1, 2);
34   insertar(L1, 3);
35   insertar(L1, 7);
36   insertar(L1, 4);
37   recorrer(L1);
38   insertar(L2, 1);
39   insertar(L2, 2);
40   insertar(L2, 3);
41   recorrer(L2);
42   Lista *L3 = diferencia(L1, L2);
43   recorrer(L3);
44   destruirLista(L1);
45   destruirLista(L2);
46   return 0;
47 }
```

Pregunta 4 (15 puntos)

Explique lo que es un puntero. Además explique para qué sirven las siguientes funciones: `malloc()`, `calloc()`, `realloc()`, `free()`. Desarrolle un programa que muestre su funcionamiento.

I) *puntero*: es un tipo de variable la cual almacena una dirección en memoria y proporciona una forma de referirse a la dirección de memoria donde está almacenado un dato.

II) `malloc`: Reserva una porción de memoria de un tamaño determinado en bytes. El prototipo de la función es `void *malloc`(tamaño en *bytes*). La función `malloc` no inicializa el espacio reservado y retorna un puntero al primer byte de la dirección de memoria reservada si tuvo éxito, en caso contrario retorna `NULL`.

III) `calloc`: Reserva n veces una porción de memoria de un tamaño determinado en *bytes*. El prototipo de la función es `void *calloc`(cantidad de veces, tamaño en *bytes*). Inicializa el bloque de memoria reservado y retorna un puntero al primer *byte* del bloque reservado.

IV) `realloc`: Cambia el tamaño del objeto apuntado por ptr al tamaño especificado por tamaño. El prototipo de la función es `void *realloc`(void *ptr, tamaño). La función realloc retorna o bien un puntero `NULL` o bien un puntero posiblemente al espacio adjudicado mudado.

v) `free`: Libera un bloque de memoria. El prototipo es `void free`(localizacion). Las llamadas para la liberación de bloques de memoria reservada previamente pueden realizarse en cualquier orden, independiente del modo o tiempo en que fueron reservados.

vi) Ejemplo de uso:

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int n = 10, i;
  int *a;
  int *b;
  a =  malloc(n * sizeof(int));
  b = calloc(n, sizeof(int));

  for(i = 0; i < n; i++)
    a[i] = 1;

  for(i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\n");

  a = realloc(a, n*2);

  for(i = 0; i < n*2; i++)
    printf("%d ", a[i]);
  printf("\n");

  for(i = 0; i < n; i++)
    printf("%d ", b[i]);
  printf("\n");

  free(a);
  free(b);

  return 0;
}
```

Pregunta 5 (15 puntos)

La criba de Eratóstenes es un algoritmo que permite encontrar los primos menores que $n \in \mathbb{N}$. Consiste en generar una lista con los número comprendidos entre 2 y n. Se comienza por p, el cual es el primer número de la lista. Luego, se marcan todos los múltiplos de p en la lista. Finalmente, p se declara primo y se procede con el siguiente número que no esté marcado, y así sucesivamente. El proceso termina cuando el cuadrado del mayor número confirmado como primo es mayor que n.

Mediante el uso de colas implemente la criba de Eratóstenes.

```
1  #include "cola.h"
2
3  bool eratostenes(Cola *c)
4  {
5    Nodo *i;
6    Info *d;
7    int j, n, p, max;
8
9    n = c->tamano;
10   pop(c); // Se elimina el 0
11   pop(c); // Se elimina el 1
12
13   while((primero(c)->datos->dato1 * primero(c)->datos->dato1) < n )
14     {
15       p = primero(c)->datos->dato1;
16       // Se imprime el primer elemento, siempre es primo.
17       printf("%d\n", primero(c)->datos->dato1);
18       max = c->tamano; // se obtiene el largo de la cola, para no recorrerla infinitamente.
19       for(j = 0; j < max; j++)
20         {
21           // Si no es multiplo de p, se debe mantener en la cola.
22           if(primero(c)->datos->dato1 %p != 0)
23             push(c, primero(c)->datos->dato1);
24           // Se saca el primer elemento para continuar analizando el resto de los números.
25           pop(c);
26         }
27     }
28   while(c->inicio != NULL) // Se imprimen el resto de los números, que siempre serán primos.
29     {
30       printf("%d\n", primero(c)->datos->dato1);
31       pop(c);
32     }
33   printf("\n");
34 }
35
36 int main(int argc, char *argv[])
37 {
38   Cola *C;
39   int i, n;
40   C = creaCola();
41
42   printf("Ingrese n: ");
43   scanf("%d", &n);
44   for(i = 0; i < n; i++)
45     push(C, i); // Se llena la cola con elementos del 0 al 20
46
47   eratostenes(C);
48
49   destruirCola(C);
50   return 0;
51 }
```